REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 07-01-2005	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* 09-09-2003 - 08-09-2004
4. TITLE AND SUBTITLE Static Analysis for Detecting Vulnerabilities in COTS		**5a. CONTRACT NUMBER** N00014-03-C-0502
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Radu Gruian and Tim Teitelbaum		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) GrammaTech, Inc.; 317 N. Aurora St., Ithaca, NY 14850		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 800 North Quincy Street Arlington, VA 22217-5660		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Unrestricted

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

20050412 018

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This is the final report of a project to study object code analysis, rewriting, and regeneration; interacting with Univ. of Wisconsin on enhancements to and applications of CodeSurfer/x86 and Weighted Moped; and supporting DOD clients in their evaluation of assurance and understanding tools developed at GrammaTech. This one-year project was a continuation of a two-year effort that involved two CIP/SW MURIs at the University of Wisconsin and Carnegie-Mellon University (CMU) managed by the Office of Naval Research (ONR), and four separately funded GrammaTech projects with similar goals.

15. SUBJECT TERMS
Static analysis binary code-rewriting code-generation model-checking Moped effect-analysis pointer analysis

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Tim Teitelbaum
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	Same as report	9	**19b. TELEPHONE NUMBER** *(include area code)* 607-273-7340

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

A Introduction

Using advanced static analysis offers the potential to find both bugs and security vulnerabilities in code. Our system, named CodeSurfer, is being applied to solving these problems. In particular, CodeSurfer is a key component of two ONR CIP/SW research projects. Researchers at the University of Wisconsin are using CodeSurfer in their project "Vulnerability and Information Flow Analysis for COTS." In addition, Carnegie Mellon University and the University of Wisconsin are using CodeSurfer's pointer analysis in their joint ONR CIP/SW project, "Static Analysis to Enhance the Power of Model Checking for Concurrent Software".

This one-year project was a continuation of a previous one-year effort (contract N00014-02-C-0188) that involved the aforementioned CIP/SW MURIs at Wisconsin and CMU, and four separately funded GrammaTech projects with similar goals. The focus of this project has been to continue to participate in the efforts of these research projects, specifically by modifying CodeSurfer to meet their infrastructure requirements for code analysis, code-rewriting, and code-regeneration; to participate in the CIP/SW project reviews; to participate with Wisconsin and CMU in their efforts to publish the results of their CIP/SW research; and to support DOD clients in their testing and evaluation of transitioned technology.

This is the final report of the project. The report summarizes work done in both years; a single dagger (†) indicates work performed solely under contract N00014-02-C-0188; a double dagger (‡) indicates work performed solely under contract N00014-03-C-0502; the absence of either dagger indicates work performed during both contracts.

B Approach

Our approach to creating an effective common infrastructure for these research projects was to start with CodeSurfer [2, 3], a tool that was originally narrowly conceived as just a program understanding system for ANSI C, and then adapt it to meet the needs of a collection of representative researchers wanting to use it for other applications. Specifically, we worked with six different efforts within two CIP/SW MURIs at the University of Wisconsin and Carnegie-Mellon University (CMU) managed by the Office of Naval Research (ONR), and four separately funded GrammaTech projects with quite similar goals:

#	Application	A. Wisconsin / CMU Projects	B. GrammaTech Projects
1	Object Code Analysis	Analysis of COTS executables for the Intel x86 family of processors (Reps, Balakrishnan)	Analysis of firmware for the Intel x86 family of processors (AFRL/Rome SBIR Phase-I and Phase-II projects "Detecting Malicious Code in Firmware")
2	Buffer-Overrun Analysis	Detection of buffer-overrun vulnerabilities in C source code (Jha, Ganapathy)	Detection of buffer-overrun vulnerabilities in C source code (AFRL/Rome SBIR Phase-I and Phase-II projects "Source Code Vulnerability Analysis")
3	Object Code Rewriting	Binary code (x86) rewriting technology for security (Jha, Miller, Giffin, and Christodorescu)	Java byte code (Jimple) rewriting technology for security (NIST SBIR Phase-I and Phase-II projects "Inline Reference Monitors for Object Code")

4	Model Checking	(a) Weighted pushdown systems (Reps, Jha) (b) Verification of properties of concurrent C programs (Clarke, Sagar)	Verification of path properties in C and C++ programs (DARPA SBIR Phase-I and Phase-II projects "Verification of Hierarchical Graph Structures")
5	Virus Detection	Detection of viruses in binary (x86) code. (Jha, Christodorescu)	(none)

C Technical Objectives

One of the goals was to create a powerful, flexible, and open toolkit for static program analysis that would support multiple programming languages, multiple computer platforms, multiple analysis algorithms, and multiple client applications. A successful design would maximize code reuse, i.e., minimize the amount of code duplication that would be required of any given user of the toolkit.

Armed with a powerful general purpose analysis infrastructure, our primary goals were (a) transferring GrammaTech technology **to** Wisconsin and CMU in support of their MURI projects, (b) transitioning research results **from** those projects back into GrammaTech, and (c) supporting early adopters of the transitioned research results in the Government.

The project was designed to be a win for each of the three parties involved:

- **Universities** would get access to high-quality, supported technology, thereby freeing them to focus on basic research. This would minimize their humdrum engineering activities, and minimize disruptions involved in day-to-day support of early adopters.
- **GrammaTech** would get access to world-class researchers and their prototypes for new cutting-edge products, as well as feedback from early adopters to guide the development of those products.
- **The Government** would avoid funding wasteful duplicate work, and accelerate transitions of basic research.

D Work Items Planned

The work items for the project were as follows:

- To design and implement a new architecture for CodeSurfer and new APIs with which users could program their static analyses; and to create two reference implementations of static analysis algorithms using the new architecture and APIs.
- To support Wisconsin's efforts to use CodeSurfer for x86 for code-rewriting and code-regeneration applications.
- To support Wisconsin's efforts to use CodeSurfer for x86 to implement effect analysis and pointer analysis for x86.
- To continue to interact with Wisconsin and Stefan Schwoon on enhancements to, and applications of Weighted Moped.
- To continue to support CMU's use of GrammaTech's Pointer Analysis Module (PAM), including making improvements to PAM, as part of our two-year effort of collaborating with CMU.

- To participate in the CIP-SW project reviews for Wisconsin and CMU, including preparation and presentation of relevant material at the reviews.
- To participate with Wisconsin and CMU in their efforts to publish the results of their CIP-SW research.
- To support DoD clients in their experimental evaluation of assurance and understanding tools developed at GrammaTech.

More specifically, we anticipated doing the following work to meet our initial technical objectives:

1. **Multi-lingual capabilities**. This work would involve
   a. *Abstracting* language-specific functionality out of CodeSurfer *per se*.
   b. *Reinstantiating* pre-existing versions of CodeSurfer using the factored language-independent services we had introduced.
   c. *Implementing* new front-ends for one or more other programming languages to demonstrate the generality of the new architecture.
2. **Builder modularization**. This work would involve
   a. *Decomposing* CodeSurfer's monolithic builder into its constituent analysis components, and creating an open API for each such component. This would allow clients of the system to program their own analyses by writing their own code using the API in whatever phase ordering or iteration schemes were most suitable for their own applications.
   b. *Implementing* new analyses making use of the newly exposed components to demonstrate the generality of the new architecture.
3. **Back-end extensions**. This work would involve
   a. *Creating* additional open APIs in the back-end to support plug-in applications.
   b. *Demonstrating* the use of those APIs by various plug-in applications.

We also anticipated the following work to meet the remaining objectives:

- **Support to MURIs**. Discussions with the MURI researchers to elicit their requirements, interactions during an iterative design cycle, delivery of new versions of software, and bug fixing in a timely manner.
- **Transition from MURIs**. Adaptation and adoption of MURI research results within GrammaTech.
- **Outreach**. Discussions with prospective early adopters, packaging and documenting results in a distributable form, delivery to early adopters, and support to them. Writing of co-authored papers.
- **Reporting**. Participation in bi-annual MURI reviews.

E Results

Our initial technical results were as follows:

1. **Multi-lingual capabilities**.

a. We developed a front-end System Development Kit (SDK) for CodeSurfer that facilitates creation of alternative front-ends for different programming languages. The SDK contains

   i. Abstract datatypes that can be used by a front-end to build and output the intermediate representations needed by the CodeSurfer builder.

   ii. The notion of a Language Module that contains all language-specific code and data needed by CodeSurfer.

   iii. Complete documentation.

   The SDK supports:

   i. A language-independent abstract-syntax-tree (AST) framework. The AST representation can be made available for use in front ends, in the dependence-graph builder, and in the back-end scripting language.

   ii. A language-independent control-flow graph (CFG) definition facility.

   iii. An abstract datatype for source-position information.

   iv. Abstract datatypes in support of pointer analysis (see PAM, below).

b. We reinstantiated our pre-existing *ad hoc* versions of CodeSurfer for C/C++ and Intel x86 using the SDK.

c. We used the SDK to implement a new version of CodeSurfer for Jimple [1], a three-address version of Java byte codes. (This work was funded by our NIST SBIR contract.)

2. **Builder modularization**.

   a. We factored CodeSurfer's pre-existing pointer analysis code into a separate Pointer Analysis Module (PAM) that can be used independently of CodeSurfer. PAM consists of

      i. An SDK for creating the intermediate representations needed by the pointer analysis engine.

      ii. The pointer analysis engine itself.

      iii. An API, termed the Pointer Analysis Data Base (PADB), for accessing the points-to results that have been computed by the pointer analysis engine.

   b. We had originally anticipated modularizing and exposing the individual analysis components of the CodeSurfer builder, thereby allowing users to instantiate different "builders" of their own choosing. However, two concerns on the part of the PIs at Wisconsin, performance and intellectual property rights, led us to implement a quite different architecture. In short, Wisconsin wanted a light-weight analysis platform for x86 binaries that could be severed from CodeSurfer altogether. Accordingly, rather than modularizing the CodeSurfer builder, we were asked to provide an effective analysis infrastructure wholly within the x86 front end. In effect, the front-end SDK, which was intended just to facilitate a client's access to the analysis capabilities of CodeSurfer's builder, became the analysis platform itself. Some features of the CodeSurfer builder, e.g., basic block analysis, were lifted from the builder and replicated in the front end. Unfortunately, this was at odds with our goal of minimizing code duplication.

3. **Back-end extensions**. Joint work by GrammaTech and Wisconsin on buffer-overrun detection led to several back-end extensions:
    a. The creation of a general-purpose browser for viewing the results of code scans.
    b. The creation of an open API for serializing CodeSurfer objects. This extension was needed to provide a persistent representation of the buffer-overrun results.

Our activities aimed at the remaining objectives were as follows:

- **Collaboration with the Wisconsin MURI**. In two of the application areas, Object Code Analysis and Buffer Overrun Analysis, the efforts at GrammaTech and Wisconsin became so tightly intertwined that it is not appropriate to describe those activities in simple unidirectional terms as "support to" or "transition from" the MURI. The efforts became true collaborations.
    - **Object Code Analysis**. Wisconsin and GrammaTech collaborated on the development of x86fe (a.k.a. "the connector"), which can be used as a standalone x86 analysis module, or as a CodeSurfer front-end. Roughly speaking, the division of labor is that Reps and Balakrishnan work on Value Set Analysis (VSA), Affine Relation Analysis (ARA), and Aggregate Structure Identification (ASI) [4, 5], and GrammaTech does the rest including
        i. *Abstract Syntax Tree (AST).* An abstract syntax tree is provided for the analyzed program.
        ii. *Control-Flow Graphs (CFG).* A control-flow graph is computed for every function.
        iii. *Use/Def Information.* Detailed use/kill/conditional-kill information is provided for every instruction.
        iv. *Register Live-Range Analysis (RLRA).* The live-ranges of registers are computed.
        v. *Basic Blocks.* Basic blocks for the entire application (including libraries) are computed.
        vi. *Call Graph.* The call graph for the entire program (including libraries) is computed.
        vii. *Support for Libraries.* A repository of pre-processed libraries is computed, from which individual procedures are demand loaded.
        viii. *Spill Regions.* Regions of code are computed in which registers are "spilled" into memory locations (e.g., "mem = eax; ...; eax = mem;").
        ix. *Register Save/Restore Instruction Pairs.* Pairs of instructions that are used to save/restore registers at a call (caller), or on entry to/exit from a procedure (callee), are computed.

x. *Port Analysis.* Pseudo-variables are created for all ports accessed by the program. Port references are determined using constant propagation.

xi. *Graph Algorithms.* Various graph algorithms and data structures are provided, such as strongly connected components (SCCs).

xii. *Support for Multiple-Entry-Point Functions.* Multiple-entry-point functions are detected and represented.

xiii. *Support for Clones.* Multiple-entry-point functions are optionally cloned and converted to single-entry-point functions.

xiv. *Support for Non-linear Functions.* Instructions that are not included in any function(s) by IDAPro are added to the function(s) that end up executing them.

xv. *Support for Import Tables.* Functions and DLLs that are imported by the program are detected.

xvi. *Register Aliases.* A map of register aliases is maintained (e.g., al -> ax -> eax).

xvii. *End-to-end Connectivity with CodeSurfer.* x86fe and CodeSurfer are kept in synch.

- o **Buffer Overrun Analysis**†. During the first year of our two-year effort, Wisconsin and GrammaTech collaborated on the development of a tool for the detection of buffer-overrun vulnerabilities in ANSI C programs. During that year, we co-authored a paper describing our joint work [6]. We tested the tool on the current version of the Washington University FTP daemon, a popular file transfer server, found 14 previously unreported overruns, and reported them to the developers.

- **Support to the MURIs.**
  - o **PAM**†. On 9/26/02, Prof. Jha requested that GrammaTech repackage CodeSurfer's pointer analysis module as a stand alone component (PAM) for use in a joint CMU/Wisconsin project involving Prof. Clarke and his student Sagar Chaki. The user manual was delivered to CMU and Wisconsin on 12/20/02. Creating PAM involved interacting closely with Mr. Chaki during the requirements, design, implementation, and deployment phases of the effort.
- **Transitions from MURIs.**
  - o **CodeSurfer/x86**. Prototype versions of CodeSurfer/x86 have been presented and delivered to numerous government and FFRDC sites. These are detailed below. The current version of CodeSurfer/x86 is described in the *CodeSurfer/x86 User Guide and Technical Reference*, which is attached as a separate volume of this final report.
  - o **Model Checking**. We transitioned the work of Reps, Schwoon, and Jha on weighted pushdown systems [8], and their prototype implementation (Weighted Moped) to GrammaTech, where we are using it as a model checking engine for the Path Inspector [7], a tool that checks sequencing properties in programs. The Path Inspector has been released as a commercial product.

- **Outreach**
  - **SPAWAR**[†]. In a separately funded effort, we worked to transition Wisconsin's research to SSC-SD (SPAWAR). We trained a SPAWAR employee to use CodeSurfer and the prototype Wisconsin/GrammaTech buffer-overrun vulnerability detector. We then used the buffer-overrun tool to analyze the GCCS-M Tactical Management Service (TMS), and found one possible overrun in this fielded program, albeit not an overrun that can be exploited to seize control of the program.
  - **Lincoln Labs**[‡]. A classified research project at MIT Lincoln Labs adopted CodeSurfer/x86 as a platform for their work on March 3, 2004. The project is directed by Dr. Robert Cunningham, and is sponsored by DARPA/ATO under their "Dynamic Quarantining of Worms" effort, Anup Ghosh, Program Manager. We provide this group with each new development version of CodeSurfer/x86 as it becomes available, and provide telephone and email support for them on an unclassified basis.
  - **IDA/CCS**[‡]. On April 21, 2004, Tim Teitelbaum and David Melski visited IDA/CCS (Bowie) and informally briefed David Cohen, David Smitley, and Jesse Draper about CodeSurfer/x86. This meeting led to a half-day visit to IDA by Tom Reps on May 19, 2004, during which he gave a formal presentation to an audience that included IDA, NSA, and ARDA representatives. This led, in turn, to an ongoing evaluation of CodeSurfer/x86 by IDA/CCS that began on June 6, 2004.
  - **Sandia**[‡]. On May 27, 2004, we hosted a visit from three researchers of Sandia National Laboratory (Steve Williams, Doug Ghormley, and Todd Jones) during which we presented the results of the Wisconsin/GrammaTech collaboration. Subsequently, we established a support arrangement with Sandia. We have delivered multiple copies of CodeSurfer/C and C++ to Sandia, and will deliver a prototype version of CodeSurfer/x86 early in 2005.
  - **NSA**[‡]. Tom Reps and Tim Teitelbaum gave several presentations to NSA, one on September 23, 2004 over the web, and a second at Ft. Meade on December 6, 2004. Discussions with NSA are ongoing.
  - **AFRL**[‡]. On April 9, 2004, Tim Teitelbaum made a two-hour presentation on CodeSurfer/x86 to Bill Wolf and John Feldman in Rome, NY.
  - **DARPA/ATO**[‡]. On May 11, 2004, Tom Reps briefed Anup Ghosh at DARPA headquarters.
  - **IRC**[‡]. In July 2004, Tom Reps briefed the Infosec Research Council in Washington. The talk was structured as a combined PowerPoint and live CodeSurfer/x86 demo. A joint Wisconsin/GrammaTech whitepaper was distributed to the attendees.
  - **Mitre**[‡]. On May 11, 2004, Tom Reps made a presentation on CodeSurfer/x86 to Mitre in Washington. This was followed by several phone conferences with Joshua D. Guttman, who heads security research at Mitre. Although he was keen to have one or more of his malicious-code researchers at Mitre begin working with CodeSurfer/x86, this transition is currently dormant.

- **CERT**[‡]**.** We were contacted by John McHugh of CERT about the possibility of using CodeSurfer/x86 in a proposed project to derive functional characterizations of x86 programs. On June 17, 2004, Tom Reps gave a web demonstration to John McHugh and David Mundie. This was followed by an extended phone conference and subsequent delivery of CodeSurfer/x86 on July 2, 2004. This transition is currently dormant to a redirection of CERT activities.

- **Reporting**
  - **MURI Reviews.** GrammaTech staff (David Melski and/or Tim Teitelbaum) participated in, and made presentations at, the Wisconsin and CMU MURI reviews in Harpers Ferry[†], Pittsburgh[†], Williamsburg[†], and Baltimore[‡].
  - **OSD.** Tim Teitelbaum and David Melski briefed[†] Andre van Tilborg at his office on July 24, 2003. Tom Reps and Tim Teitelbaum briefed[‡] Andre van Tilborg and Steve King over the web on June 7, 2004.
  - **ONR**[‡]**.** Tim Teitelbaum (and Tom Reps by phone) briefed Gary Toth at ONR on April 29, 2004.

F Conclusions

Most research projects must spend a great deal of time developing infrastructure before they can begin addressing interesting research questions. Projects at both Wisconsin and CMU were able to reach their goals more efficiently by building on top of infrastructure provided by GrammaTech, and by being involved in a feedback loop that ultimately tailored that infrastructure to better suit their research needs. At the same time, GrammaTech was able to transition research results from Wisconsin into usable technology that helps solve real world problems.

References

1. *Soot: a Java Optimization Framework.* McGill University.
2. *CodeSurfer User Guide and Reference Manual.* 2004, Ithaca, NY: GrammaTech, Inc.
3. Anderson, P. and T. Teitelbaum, *Software Inspection using CodeSurfer.* In *WISE 01, Workshop on Inspection in Software Engineering.* 2001. Paris.
4. Balakrishnan, G. and T. Reps, *Analyzing memory accesses in x86 binary executables.* 2003, Computer Sciences Department, University of Wisconsin, Madison, WI TR-1486.
5. Balakrishnan, G. and T. Reps, *Analyzing Memory Accesses in x86 Executables.* In *International Conference on Compiler Construction.* 2004.
6. Ganapathy, V., *et al.*, *Buffer Overrun Detection using Linear Programming and Static Analysis.* In *10th ACM Conference on Computer and Communications Security.* 2003. Washington, DC.
7. GrammaTech, *The Path Inspector.* 2004, http://www.grammatech.com/products/codesurfer/overview_pi.html.
8. Reps, T., S. Schwoon, and S. Jha, *Weighted pushdown systems and their application to interprocedural dataflow analysis.* In *10th Int. Static Analysis Symp.* 2003. San Diego, CA.